Egocentric Storylines for Visual Analysis of Large Dynamic Graphs

Chris W. Muelder*, Tarik Crnovrsanin*, Arnaud Sallaberry†, and Kwan-Liu Ma*
*University of California at Davis
Davis, CA, USA
muelder/ma@cs.ucdavis.edu, tacrnovrsanin@ucdavis.edu
†LIRMM, Universitè Paul Valéry Montpellier 3
Montpellier, France
arnaud.sallaberry@lirmm.fr

***Abstract*—Large dynamic graphs occur in many fields. While overviews are often used to provide summaries of the overall structure of the graph, they become less useful as data size increases. Often analysts want to focus on a specific part of the data according to domain knowledge, which is best suited by a bottom-up approach. This paper presents an egocentric, bottom-up method to exploring a large dynamic network using a storyline representation to summarise localized behavior of the network over time.**

Keywords*—*information visualization; dynamic graphs; storylines; egocentric views;

I. Introduction

Relational data, consisting of a set of entities and a network of relationships between them, is one of the primary classes of information (some other examples being spatial data, n-dimensional data, or text data). This kind of data turns up in many different disciplines, including sociology, biology, engineering, computer networks, and many more. Analysis of these networks can lead to important insights, but as with other kinds of data, the scale of network data is readily increasing. Facebook, for instance, now has over a billion users forming an incredibly complex network of friendships, communication, and more. As such networks become increasingly larger and more complex, reasoning social dynamics via simple statistics or even many traditional visalization techniques are no longer feasible options.

Many visualizations follow the top-down information visualization mantra of "Overview first, Zoom and filter, Details on demand" [35]. However, as the scale increases, overviews become less useful; since the capacity of the display (and the human eye) has an upper bound, as the data size increases, an overview will convey a diminishing percentage of the total information in the dataset. This is particularly problematic if the analyst already knows what part of the network they want to focus on, as overviews tend to abstract away the specific details that the analyst is looking for. As an alternative, many visualization approaches take a bottom-up approach of "Search, Show context, Expand on demand" [38]. While these approaches depend on domain knowledge or statistics to provide a starting point, they are often helpful for large data analysis, as the amount of information provided to the user at any time can be controlled independently from the size of the data set.

Another challenge is that many graphs are not static; data collected in real world applications are often intrinsically time-varying. For instance, in a social network over time, new friendships can be made, or old friendships lost. While the problem of visualizing static networks has been studied quite extensively, work on dynamic network visualization is still in its infancy. Most existing works focus on the layout stability of time-varying node-link diagrams, either shown as animation or small multiples. However, for large data, animation can take a long time to watch, and small multiples reduce the already limited screen space. A few works investigate methods of creating time-lines for dynamic graphs to statically summarize the evolution of dynamic graphs. However, these methods suffer from the same overview limitations mentioned above.

This paper presents a method for visualizing a dynamic graph that combines both bottom-up network exploration and a static timeline representation. The result is an egocentric storyline based visualization that summarizes the behavior of the local network surrounding user-selected foci. This approach was also designed as a streaming algorithm, making it viable for online dynamic graph data.

II. Related Work

A common method for visualizing dynamic graphs is to animate the transitions between time-steps [25], [10], [12], [15], [3], [13]. This approach yields dynamic visualization with nodes appearing, disappearing and moving to produce readable layout for each time-step. Alternatively, multiple time-steps can be staticly placed next to each other using "Small Multiples" [37]. This eases the comparison of distant time-steps but the area devoted for each time-step is small and this reduces the readability of each graph. An empirical study to compare the advantages and drawbacks of these approaches ("Animation" vs. "Small Multiples") has been performed by Archambault *et al.* [1].

A major issue for both methods is to ensure the stability of the layout [19], [13], [5], [17]. A stable layout helps preserve the user's mental map as there is less movement between time-steps, but sacrifices quality in terms of readability for later time-steps as their layout depends on previous time-steps. Many experiments has been proposed to examine the effect of preserving the mental map in dynamic graphs visualization [29], [32], [30]. The results of [30] were quite surprising because the most effective visualizations were the extreme ones,

i.e. the ones with very low or high mental map preservation: visualizations with medium preservation performed less well. However, most of these works do not demonstrate scalability. The work of Sallaberry *et al.* [33] is one of the few dynamic graph layout approaches that has been shown to scale to tens or hundreds of thousands of nodes, but it still relies on visualizing the entire network at once, so even it can not handle the millions or billions of nodes that can occur in modern graphs.

Hu *et al.* [17] proposed a method based on a geographical metaphor to visualize a summary of clustered dynamic graphs. They also use a clustering approach similar to ours, though it sacrifices some local quality for calculation efficiency and additional temporal stability.

An alternate visualization approach for dealing with dynamic large directed graphs is to directly represent time as an axis. In the work of Burch *et al.* [6], vertices are ordered and positioned on several vertical parallel lines, and directed edges connect these vertices from left to right. Each time-step's graph is thus displayed between two consecutive vertical axes.

Storyline visualizations have become popular in recent years for showing evolution of clusterings [22], [27], [18], [31], [9], [36]. Most of these works reference hand-drawn diagrams such as XKCD's movie narrative charts [22] as inspiration, in which entities are represented as lines which move together when in the same group and separate when they are not. "Plotweaver" [28] is a tool to aid in semi-automatic generation of storyline plots, but it still requires a lot of user interaction. The works of Ogawa et al. [27] and Tanahashi et el. [36] aim to automate the process. However, producing good results with these algorithms is computationally expensive, so they do not scale well to large data [36]. Working with larger data can make the lines start to blend together into larger flow-like structures similar to sankey diagrams [9], [26]. To apply storyline techniques to dynamic graphs, an intermediary step of dynamic clustering must be derived [31], [33]. These approaches use bins to more efficiently lay out the storylines.

Since overviews are less useful for enormous networks, researchers have introduced several bottom-up techniques. These approaches start from a single selected node and its immediate context. Additional relevant nodes and connections are revealed only on demand, based on graph structure or specialized degree-of-interest functions. Moscovich et al. [21] designed two intuitive interaction techniques called "Link Sliding" and "Bring & Go" for navigating large networks. Heer and Boyd [16] presented a visualization method which only shows a focus node's neighboring nodes up to a certain level. Similarly, Elmqvist and Fekete [11] described a bottom-up system based on hierarchy traversal methods, including above traversal, below traversal, and level traversal. These methods are useful when the inherent graph structure is more important than other properties for the task at hand. For other applications, where node/edge attributes are the focus of analysis, researchers create specialized degree-of-interest (DOI) functions. Furnas [14] introduced a DOI function to evaluate the importance of a selected node based on distance and a priori interest. Van Ham and Perer [38] extended this function to operate on embedded attributes and graph topology, as well as user-generated search actions. Their system can then suggest nodes with the highest degrees of interest for users to explore. Crnovrsanin et al. [8] combine this concept with an interaction history based importance similar to Amazon's item-to-item collaborative filtering [20]. Our approach works with dynamic graphs, so we can use temporal history as well as interaction history.

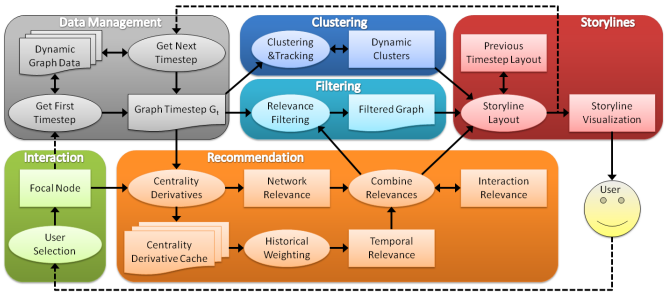


Fig. 1. *Our overall process.* The user selects a focal node (in green), the system loads the graph data one timestep at a time (grey), clusters the time step (blue), derives relevence values (orange), uses the relevence values to filter the timestep (cyan), and appends the timestep to the storylines (red).

III. Approach

Storylines depict the evolution of dynamic clusters over time. So to use storylines to depict a dynamic network, the first step in our approach is to derive such a dynamic clustering. Once we have this clustering, we want to be able to focus on small portions of the network at a time. We employ an egocentric approach to start with a single focal point and use a recommendation algorithm to provide the local context. Once we have this localized subset of the data, we employ an efficient storyline layout algorithm to create the resulting visualization. These storylines can then be explored either through expansion via user selection of new foci or temporally via user selection of time steps to view in more detail.

A. Dynamic Graph Clustering

A dynamic graph can be defined formally as an agglomerate graph $G = (V, E)$ and an ordered sequence of subgraphs $S = \{G_1 = (V_1, E_1), G_2 = (V_2, E_2), ..., G_k = (V_k, E_k)\}$ where each G_t is the subgraph of G at time t. $V, V_1, V_2, ..., V_k$ are finite and non-disjointed sets of nodes, $E, E_1, E_2, ..., E_k$ are finite and non-disjointed sets of edges such that $V = V_1 \cup V_2 \cup ... \cup V_k$ and $E = E_1 \cup E_2 \cup ... \cup E_k$. What we need is to create a time-varying clustering, *i.e.* a set of clusters evolving over time. The clustering method we describe here is a two step algorithm. The first step consists of partitioning the nodes for each time step independently. Then, we associate these clusters through time to derive time-varying clusters.

1) Time-step Clusterings: Finding a partition of the nodes of a static graph according to its structure is a well studied problem. Schaeffer has published a good overview of graph clustering methods[34]. For our approach, we need to cluster a dynamic graph, which is a less studied problem. We do this by first finding a partition for each time step, *i.e.* a set of clusterings $C = \{C_1, C_2, ..., C_k\}$ where $C_t = \{c_1^t, c_2^t, ..., c_{lt}^t\}$ is a partition of the nodes V_t of G_t. In this paper, we call each C_t a "time-step clustering" where c_i^t is the "time-step cluster" i at time t, and $c_i^t \subseteq V_t$ for $1 \leq i \leq lt$, $V_t = c_1^t \cup c_2^t \cup ...c_l^t$ and $c_i^t \cap c_j^t = \emptyset$ for $i \neq j$,

Our algorithm is based on the so-called modularity function [23]. It represents the sum of the number of edges linking nodes of the same clusters minus the expected such sum if edges were distributed at random. For a graph $G_t = (V_t, E_t)$ and a partition C_t of its nodes, the modularity $Q(C_t)$ is defined by:

$$Q(C_t) = \frac{1}{2|E_t|} \sum_{u,v \in V_t} \left[A_{uv} - \frac{k_u k_v}{2|E_t|} \right] \delta(c^t(u), c^t(v))$$

where $|E_t|$ is the number of edges, A_{uv} is 1 if there is an edge between u and v and 0 otherwise, $k_u = \sum_v A_{uv}$ is the number of edges attached to u, $c^t(u)$ is the time-step cluster of C_t containing u, $\delta(c^t(u), c^t(v))$ is 1 if $c^t(u) = c^t(v)$ and 0 otherwise.

A partition that maximizes this function helps to discover clusters of densely connected communities. Moreover, as shown by Noack [24], optimizing the modularity is the same as optimizing an energy function in graph layout. This equivalence implies that our layout based on such a clustering algorithm yields a good representation of the graph.

The problem of finding a partition that maximizes the modularity is hard, and the corresponding decision problem is NP-complete [4]. We use the heuristic proposed by Blondel *et al.* [2], which works well in terms of both the quality of the results and the computation time. Initially, each node belongs to its own cluster. Then pairs of clusters are recursively merged such that the modularity of the partitioning increases. If two possible merges involve the same cluster, the merge that improves the modularity the most is performed.

2) Cluster Tracking: We define a time-varying clustering of a dynamic graph G as a set of time-varying clusters $VC = \{VC_1, VC_2, ..., VC_l\}$. Each of these time-varying clusters is an ordered sequence $VC_i = \{vc_i^1, vc_i^2, ..., vc_i^k\}$ where k is the number of time steps and each vc_i^t is a subset of the vertices V_t at time t. That is, each time-varying cluster VC_i is a cluster whose membership can evolve over time, where vc_i^t represents the set of nodes in the cluster i at time t. As the number of clusters can change between timesteps, not every cluster exists at every timestep, so the total number of time-varying clusters l can be larger than the number of time step clusters at any time step.

Our overall approach is to compare the time-step clusters pairwise between neighboring time-steps and putting the most similar time-step clusters into the same time-varying cluster. We start from an empty set VC of time-varying clusters and we create a time-varying cluster VC_i for each time-step cluster c_i^1 of the first time-step clustering C_1. The set of nodes of these time-varying clusters VC_i at time 1 are initialized with the time-step clusters c_i^1: $vc_i^1 \leftarrow c_i^1$.

Then, starting with this partition of the graph at time 1, for each vc_i^2 we search for the time-step cluster of C_2 that is the most similar to vc_i^1. Let c_a^2 be such a cluster, then $vc_i^2 \leftarrow c_a^2$. If no similar cluster can be found in C_2, then vc_i^2 is an empty set (i.e. the time-varying cluster has disappeared at time-step 2). If there is a time-step cluster c_a^2 in C_2 that cannot be associated with a vc_i^2, then a new time-varying cluster VC_b is created with $vc_b^1 \leftarrow \emptyset$ and $vc_b^2 \leftarrow c_a^2$. We iterate this process for each time-step.

The crux of this algorithm is how to decide which time-step cluster of C_t is the most similar to a cluster of C_{t-1}. The solution we use is based on a similarity function between time-step clusters of C_{t-1} and clusters of C_t. Results of this function can be stored in a matrix M such that M_{ij} denotes the similarity between c_i^{t-1} and c_j^t. Starting from the highest value of this matrix and the corresponding clusters c_i^{t-1} and c_j^t, we assign c_j^t to the time t of the dynamic cluster VC_i that contains the cluster c_i^{t-1} at the time $t-1$. Then we do the same for the second highest value of the matrix and so on. Any values corresponding to pairs of time-step clusters that have already been assigned a dynamic cluster are skipped, as a better match was already found earlier in the algorithm. This process ends when there are no more time-step clusters or when the highest similarity value is under a given $threshold$. Finally, any remaining time-step clusters become new dynamic clusters.

In our implementation, we use the Jaccard index to compute the similarities. For two clusters c_i^{t-1} and c_j^t, this is defined by the equation $|c_i^{t-1} \cap c_j^t| / |c_i^{t-1} \cup c_j^t|$. There are two main advantages in using this metric. First it takes into account the number of shared nodes as well as the total number of nodes, which guarantees homogeneity between consecutive steps of a time-varying cluster. Secondly it returns a value normalized between 0 and 1 which is helpful for empirically defining a $threshold$.

B. Egocentric Context Recommendation

Rather than trying to show all clusters simultaneously, we opt for a bottom-up technique. In this approach, the user selects a single node of interest (the 'ego'), and the system determines and presents what is relevant with respect to the selection. We do this by applying a recommendation algorithm to the network. Similar to the Degree Of Interest (DOI) functions of Furnas [14], Van Ham and Perer [38] or Crnovrsanin et al. [8], we use a weighted combination of several relevence functions. We combine a network relevence metric with a temporal relevence history and an interaction relevance history.

1) Network Relevence: Many existing works use centrality sensitivity calculations to derive relative network-based importance [7], [8]. However, at the scale of data we were analyzing, we found that the standard centrality algorithms were too expensive to calculate dynamically, and the sensitivities used too much storage space to be viable, as they consist of an entire $|V|^2$ sized matrix for each time step. So instead, we use graph distances directly. Since we have a single focal vertex, these distances can be computed very efficiently using Dijkstra's algorithm. Then, we define the network relevance simply as the inverse of the graph distance to the focal node. That is:

$$NR_{i,j,t} = \frac{1}{d_{i,j}}$$

2) Temporal Relevence: As in dynamic graph layouts, it is important that the storylines be stable over time [36]. Thus, rather than computing the overall relevance exclusively on the current time step being worked with, we want to incorporate

previous timesteps into the relevance metric. However, it would be wasteful to recalculate the relevances of previous timesteps, so we cache them for a user controllable number of previous timesteps. That is, for each of the k time steps before the current time step t (i.e.: $G_{t-k}, G_{t-k+1}, ..., G_{t-1}$), we save the network relevances between each pair of nodes. So for each pair of nodes i and j in the current time step, we have an array of the network relevances from the previous k timesteps $\{NR_{i,j,t-k}, NR_{i,j,t-k+1}, ..., NR_{i,j,t-1}\}$. In order to combine these into a single temporal relevance value, one could simply average them together, but this would not produce a very smooth/stable value, as outliers would have a strong effect for many time steps. Instead, we compute a triangular weighted average by applying a linear weight $w_{t-a} = \frac{k-a+1}{k}$ for $1 \leq a \leq k$. Thus, we compute the overall temporal relevance as:

$$TR_{i,j,t} = \frac{NR_{i,j,t-k} * w_{t-k} + ... + NR_{i,j,t-1} * w_{t-1}}{w_{t-k} + ... + w_{t-1}}$$

This creates a sliding window which allows the effect of old values to fade out smoothly. Higher order window functions could be applied (e.g. quadratic or cosine weights), but we found the linear window to be sufficient.

3) Interaction Relevence: In order to aid the user in keeping mental track during their exploration, we want to preserve some of the context of their previous steps. Each time the user selects a new focal node, we want to preserve the importance of the previous selection to maintain context. One way to do this would be to cache the relevences of the most recent selections, and weight them to have them fade out as more selections are made, similarly to temporal relevance. However, if we cache the combined relevence, then each cached value would already include the previous timestep, weighted by the combination function, and recursively, every time step before that. As long as the weighting is less than 1, this will produce an exponential fall off. Thus, we calculate the interaction relevance as:

$$IR_{i,j,t} = CR_{k,j,t}$$

where $CR_{k,j,t}$ is the combined relevence at time t of node j with respect to the previously selected node k.

4) Combined Relevence Metrics: We combine these 3 metrics with a user controllable linear summation:

$$CR_{i,j,t} = \alpha * NR_{i,j,t} + \beta * TR_{i,j,t} + \gamma * IR_{i,j,t}$$

where α, β, γ are controlled via the interface. However, to seperate the layout from the filtering computation, we use 2 sets of constants to define 2 combined metrics: one used for filtering and one used for layout. This allows for more flexible control: e.g. the user can set it so that the filtering uses $IR_{i,j,t}$ but not the layout, so that previous selections are still included in the plot, but do not interfere with the layout of the current selection.

C. Streaming Storyline Layout

With a good layout, storylines have been shown to be effective for visualizing evolution of dynamic clusters. But computing a good layout can be costly; e.g. Tanahashi's layout uses a genetic algorithm to find a global optimization [36], which produces good results but takes a long time. In order to lay out large, streaming data, we need a more efficient

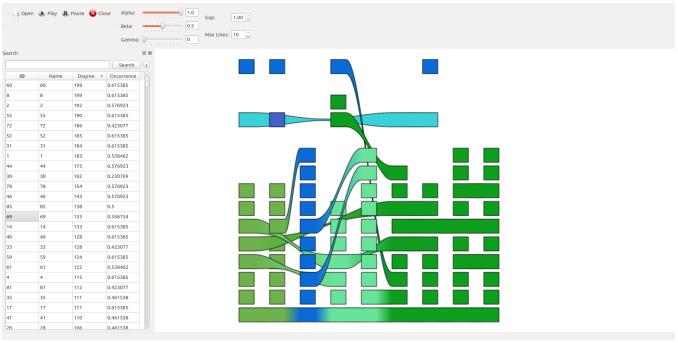


Fig. 2. An example of our UI. One node is searched for and selected from a list. Then the relevant nodes are plotted in the storyline view, laid out with the selected node at the bottom and more relevant neighbors placed closer. The lines are colored either according to the clustering or node id.

approach. Since we have a focal point and have computed relevance metrics with respect to this focal point, we can use the nodes' relevences to give semantic meaning to the ordering and placement of the storylines. Towards these goal, we employ a greedy layout algorithm, which makes a best-effort to quickly lay out each time step one at a time while using the relevence information.

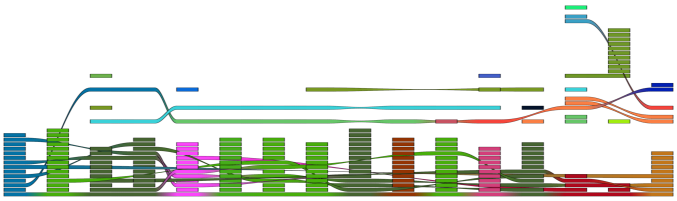
Similarly to the works of Sallaberry et al. [33] and Reda et al. [31] our layout algorithm is a bin-based algorithm. For each cluster in the current time step, we use the average relevence of the cluster's nodes and the cluster's assigned slot from the previous time step (if it existed in the previous time step) to determine the optimal bin to assign the cluster to. If the optimal bin is already occupied, then the algorithm assigns the cluster to the next available bin instead. Once assigned, the cluster does not get reassiged to a different bin, so the order of insertion is important. Stability is more important than rigorous ordering, so priority is given to clusters that existed in the previous time steps. Within both groups (pre-existing and new clusters), the clusters are inserted in order of importance according to the relevence metric.

Once the clusters have been arranged into bins, we need to arrange the individual lines within the cluster. One simple way to do this would be to simply order them by relevance. However, this can yield many gaps between lines and does not offer any guarantee of stability. Instead, we carry over any placement from the previous time step, insert any remaining nodes ordered by relevence, and finally remove any gaps if there are any. In this manner, the ordering will not be perfectly by relevence, but the layout will be far more stable.

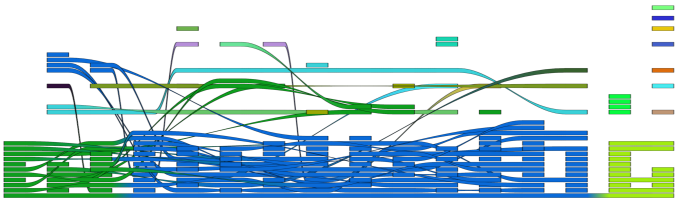
Since our layout is a greedy approach, it can get stuck in local minima (e.g. when the optimal bin is already occupied). While this does sacrifice some level of quality compared to a global optimization, we found that the results are still sufficient for our purposes.

D. Interactive Exploration

Any bottom up approach needs to provide the user a way to search for and select a starting point. We provide a simple table with a search/filter mechanic that can be sorted or filtered on arbitrary data attributes. This way, if the user has a starting point in mind, they can find it quickly. Alternately, the user



(a) Selecting Node 15



(b) Selecting Node 55

Fig. 3. MIT reality mining dataset. Selecting various nodes reveals differing patterns. Node 15 changes clusters almost every timestep as the similarity between clusters fell under the clustering threshold. Node 55 is more stable, but also shifts cluster membership as other nodes leave or join its cluster.

can start with a more generic search, such as starting from nodes of high degree. Figure 2 shows an example of such a selection via our user interface.

Once the user makes an initial selection, a storyline plot is generated. From this plot, the user can select any of the plotted lines to refocus the plot on the selected line. In this manner, the user can expand the view to include neighboring nodes.

IV. Results

We demonstrate our approach on several data sets. First, we apply it to the phone records from the MIT reality mining dataset, aggregated per day ($|V_t| \approx [750, 1400], |E_t| \approx [1500, 2500]$). This data set contains many transient nodes due to external phone calls, so it is quite noisy at times. But our approach reveals some interesting patterns. Figure 3 shows some examples. In both images, there are many lines that exist for only one timestep. Node 15 (Figure 3(a)) is clustered with many of these, so it does not have a consistant clustering over time. Node 55 (Figure 3(b)) has a more consistent set of neighbors, and hence a more consistant clustering over time. The autonomous systems (AS) of the Internet provide a much larger and more interesting test case ($|V_t| \approx [12k, 33k], |E_t| \approx [23k, 71k]$). Figure 4 shows several examples from a year and a half subset of the available time range. AS8307 (Figure 4(a) is clustered with a number of other stable nodes, but their cluster changes cluster ID a few times as it gradually incorporates nodes from the other cluster shown in the plot. AS9695 (Figure 4(b)) on the other hand has many chaotic neighbors in its cluster, but little interaction with other clusters. Larger autonomous systems such as AS174 or AS3356 (Figures 4(c) and 4(d)) interact with many other autonomous systems, and often serve as bridges between clusters. Thus, their cluster assignment is somewhat less stable, and they are tightly connected to numerous other clusters. Another interesting pattern that occurs is when a node starts as a member of one cluster, then splits off to form a new cluster, such as AS3707 or AS10189 (Figures 4(e) and 4(f)). Extending our approach to a decade's worth of time steps of the Internet dataset yields a large and more chaotic storyline plot, but which still has patterns similar to those in the smaller set, as shown in Figure 5. While screenspace starts to be an issue as the number of timesteps grows, our approach streams each time step, so it scales well computationally.

V. Future Work

While our approach works through the data one timestep at a time, it is still not truly streaming, where individual changes would be streamed in and handled one at a time. The clustering and relevance calculation can be computationally expensive, and it would be beneficial for them to be replaced with alternatives that could handle incremental changes. The heuristic approach of Hu et al might be a good place to start [17]. Then the approach might even be efficient enough for real-time analyses. The other direction that could be pursued is to improve the stability of the clustering itself. While this could detract from the localized cluster quality, it would improve the readability of the visualization overall. The system could also use a detailed view of the underlying network itself. As in the work of Sallaberry et al. [33], the storyline plot can be used to define a 2-dimensional graph layout via a space filling curve. Since the storylines guarantee clustering and aim for stability, they would define a stable and useful dynamic layout.

VI. Conclusion

We have presented a new, bottom-up approach to large dynamic graph visualization. While overviews are still helpful to convey the overall structure of a large network, they can be computationally costly and can either be overly cluttered or overly simplified. Bottom-up approaches such as the one presented here enable analysts to more directly impart domain knowledge into the analytics process and more practically explore the specific regions of the network of interest. We have demonstrated the effectiveness of our approach by applying it to several fairly large networks, but the method was designed with even larger future networks in mind. Techniques such as ours will become even more vital analysis tools as the size of data is ever increasing.

Acknowledgment

This work was sponsored in part by the U.S. National Science Foundation through grants CCF-0938114 and CCF-0811422, and also by the U.S. Department of Energy through the SciDAC program with Agreement No. DE-FC02-06ER25777 and DE-FC02-12ER26072.

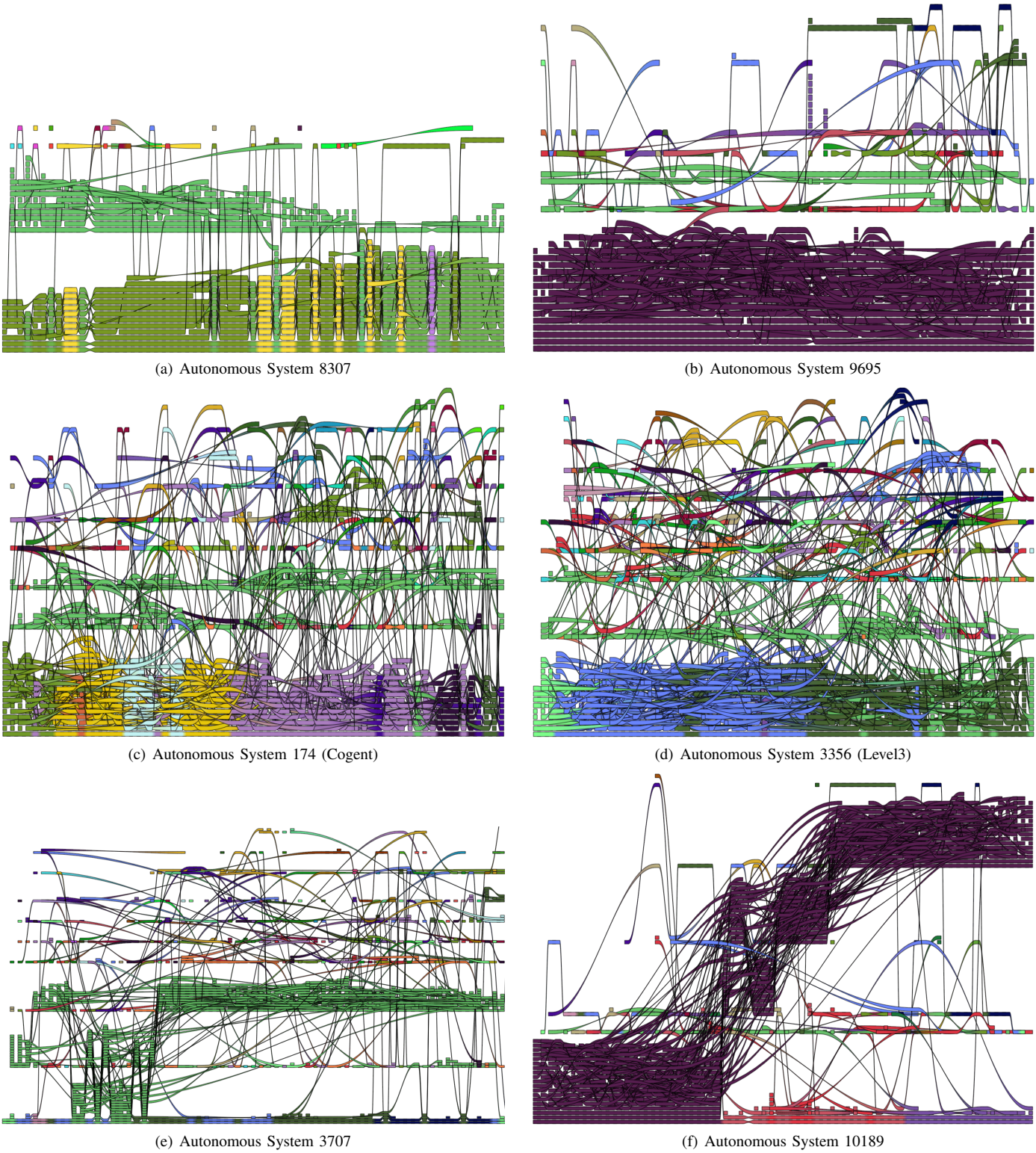
References

[1] Daniel Archambault, Helen C. Purchase, and Bruno Pinaud. Animation, small multiples, and the effect of mental map preservation in dynamic graphs. *IEEE TVCG*, 17(4):539–552, 2011.

[2] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[3] Krists Boitmanis, Ulrik Brandes, and Christian Pich. Visualizing internet evolution on the autonomous systems level. In *Proceedings of the International Symposium on Graph Drawing (GD'07)*, volume 4875 of *LNCS*, pages 365–376. Springer, 2008.

[4] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *arxiv.org/abs/physics/0608255*, 2006.



(a) Autonomous System 8307

(b) Autonomous System 9695

(c) Autonomous System 174 (Cogent)

(d) Autonomous System 3356 (Level3)

(e) Autonomous System 3707

(f) Autonomous System 10189

Fig. 4. The Internet dataset produces much more interesting patterns. Some nodes participate in very slowly evolving yet very stable clusters (a). Others have a consistant core network and connections to neighboring clusters while neighbors fluctuate (b). High visibility autonomous systems such as those operated by Cogent or Level3 have even more local flux, with an evolving base cluster yet strong ties to many external clusters (c,d). Some other nodes start out tightly connected to existing clusters before branching off to start their own subnetworks (e,f).

Fig. 5. Internet dataset. When viewed over a decade's worth of time, the details can become cluttered, but trends similar to those mentioned before are visible.

[5] Ulrik Brandes and Martin Mader. A quantitative comparison of stress-minimization approaches for offline dynamic graph drawing. In *Proceedings of the International Symposium on Graph Drawing (GD'11)*, volume 7034 of *LNCS*, pages 99–110. Springer, 1012.

[6] Michael Burch, Corinna Vehlow, Fabian Beck, Stephan Diehl, and Daniel Weiskopf. Parallel edge splatting for scalable dynamic graph visualization. *IEEE TVCG*, 17(12):2344–2353, 2011.

[7] Carlos D. Correa, Tarik Crnovrsanin, and Kwan-Liu Ma. Visual resoning about social networks using centrality sensitivity. *IEEE TVCG*, 18(1):106–120, 2012.

[8] Tarik Crnovrsanin, Isaac Liao, Yingcai Wuy, and Kwan-Liu Ma. Visual recommendations for network navigation. pages 1081–1090, 2011.

[9] Weiwei Cui, Shixia Liu, Li Tan, Conglei Shi, Yangqiu Song, Zekai Gao, Huamin Qu, and Xin Tong. Textflow: Towards better understanding of evolving topics in text. *IEEE TVCG*, 17(12):2412–2421, 2011.

[10] Stephan Diehl and Carsten Görg. Graphs, they are changing. In *Proceedings of the International Symposium on Graph Drawing (GD'02)*, volume 2528 of *LNCS*, pages 23–30. Springer, 2002.

[11] Niklas Elmqvist and Jean-Daniel Fekete. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE TVCG*, 16(3):439–454, 2009.

[12] Cesim Erten, Philip J. Harding, Stephen G. Kobourov, Kevin Wampler, and Gary V. Yee. GraphAEL: Graph animations with evolving layouts. In *Proceedings of the International Symposium on Graph Drawing (GD'03)*, volume 2912 of *LNCS*, pages 98–110. Springer, 2004.

[13] Y. Frishman and A. Tal. Online dynamic graph drawing. *IEEE TVCG*, 14(4):727–740, 2008.

[14] G W Furnas. Generalized fisheye views. In *Human Factors in Computing Systems CHI*, pages 16–23, 1986.

[15] Carsten Görg, Peter Birke, Mathias Pohl, and Stephan Diehl. Dynamic graph drawing of sequences of orthogonal and hierarchical graphs. In *Proceedings of the International Symposium on Graph Drawing (GD'04)*, volume 3383 of *LNCS*, pages 228–238. Springer, 2004.

[16] Jeffrey Heer and Danah Boyd. Vizster: visualizing online social networks. In *IEEE Symposium on Information Visualization*, pages 32–39, 2005.

[17] Yifan Hu, Stephen G. Kobourov, and Sankar Veeramoni. Embedding, clustering and coloring for dynamic maps. In *Proceedings of the 5th IEEE Pacific Visualization Symposium*, pages 33–40, 2012.

[18] Nam Wook Kim, Stuart K. Card, and Jeffrey Heer. Tracing genealogical data with timenets. In *Proceedings of the International Conference on Advanced Visual Interfaces*, AVI '10, pages 241–248, New York, NY, USA, 2010. ACM.

[19] Gautam Kumar and Michael Garland. Visual exploration of complex time-varying graphs. *IEEE TVCG*, 12(5):805–812, 2006.

[20] Greg Linden, Brent Smith, and Jeremy York. Amazon.com Recommendations: Item-to-Item Collaborative Filtering. *IEEE Internet Computing*, 7:76–80, 2003.

[21] Tomer Moscovich, Fanny Chevalier, Nathalie Henry, Emmanuel Pietriga, and Jean-Daniel Fekete. Topology-Aware Navigation in Large Networks. In *SIGCHI conference on Human Factors in computing systems*, pages 2319–2328, 2009.

[22] Xkcd #657: Movie narrative charts. http://xkcd.com/657, dec. 2009.

[23] M. E. J. Newman and M. Girvan. Graph clustering. *Physical Review E*, 69(026113), 2004.

[24] A. Noack. Modularity clustering is force-directed layout. *CoRR*, abs/0807.4052, 2008.

[25] Stephen C. North. Incremental layout in DynaDAG. In *Proceedings of the International Symposium on Graph Drawing (GD'95)*, volume 1027 of *LNCS*, pages 409–418. Springer, 1996.

[26] Michael Ogawa, Kwan liu Ma, Christian Bird, Premkumar Devanbu, and Alex Gourley. Visualizing social interaction in open source software projects. In *Proceeding of the 2007 Asia-Pacific Symposium on Visualisation*, pages 25–32. IEEE Computer Society, 2007.

[27] Michael Ogawa and Kwan-Liu Ma. Software evolution storylines. In *Proceedings of the 5th international symposium on Software visualization*, SOFTVIS '10, pages 35–42, New York, NY, USA, 2010. ACM.

[28] V. ogievetsky. plotweaver xkcd/657 creation tool. https://graphics.stanford.edu/wikis/cs448b-09-fall/FPOgievetskyVadim, march 2009.

[29] H.C. Purchase, E. Hoggan, and C. Görg. How important is the "mental map"? - an empirical investigation of a dynamic graph layout algorithm. In *Proceedings of the International Symposium on Graph Drawing (GD'06)*, volume 4372 of *LNCS*, pages 184–195. Springer, 2007.

[30] H.C. Purchase and A. Samra. Extremes are better: Investigating mental map preservation in dynamic graphs. In *Proceedings of the 5th International Conference on Diagrammatic Representation and Inference (Diagrams 2008)*, volume 5223 of *LNCS*, pages 60–73. Springer, 2008.

[31] Khairi Reda, Chayant Tantipathananandh, Andrew Johnson, Jason Leigh, and Tanya Berger-Wolf. Visualizing the evolution of community structures in dynamic social networks. *Computer Graphics Forum*, 30(3):1061–1070, 2011.

[32] P. Saffrey and H.C. Purchase. The "mental map" versus "static aesthetic" compromise in dynamic graphs: A user study. In *Proceedings of the 9th Australasian User Interface Conference (AUIC2008)*, pages 85–93, 2008.

[33] Arnaud Sallaberry, Chris Muelder, and Kwan-Liu Ma. Clustering, visualizing, and navigating for large dynamic graphs. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing*, volume 7704 of *Lecture Notes in Computer Science*, pages 487–498. Springer, 2012.

[34] S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[35] Ben Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. In *IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[36] Y. Tanahashi and Kwan-Liu Ma. Design considerations for optimizing storyline visualizations. *IEEE TVCG*, 18(12):2679–2688, 2012.

[37] Edward R. Tufte. *Envisionning Information*. Graphics Press, 1990.

[38] Frank van Ham and Adam Perer. Search, Show Context, Expand on Demand: Supporting Large Graph Exploration with Degree-of-Interest. *IEEE TVCG*, 15(6):953–960, 2009.